Architectural Tactics to Support Rapid and Agile Stability

Felix Bachmann, SEI
Robert L. Nord, SEI
Ipek Ozkaya, SEI

Abstract. The essence of stability in software development is the ability to produce quality software with infrastructure that will meet long-term business goals. The essence of rapid and agile development is the ability to deliver capabilities quickly based on customer priorities. Stability often requires cross-functional analysis and infrastructure support that will build foundational technology for the capabilities to stand on, which takes time and resources. But today's organizations must attend to both agility and enduring design. This article presents three tactics that support rapid and agile stability: aligning feature-based development and system decomposition, creating an architectural runway, and using matrix teams.

Today's organizations must design, develop, deploy, and sustain systems for several decades and manage system and software engineering challenges simultaneously; neither agility nor attention to enduring design can be dispensed with [1]. Systems developed at such a scale go through several funding and review cycles and are typically meant to operate for several decades, so longevity and stability are key goals. Shrinking defense budgets and continued demand for new capabilities add pressure to use rapid and agile development and deployment. All software-intensive systems relevant to the DoD fit this description due to the existing acquisition processes and the need to support the systems for extended periods of time in the field in order to manage costs.

This article presents the lessons we learned from our interactions with teams and individuals in the roles of Scrum master, developer, project manager, and architect on projects from organizations that develop embedded real-time software or cyber-physical systems. We observed that the following symptoms surface from the lack of stability to sustain rapid and agile software development:

- Scrum teams, product development teams, component teams, or feature teams spend almost all of their time fixing defects, and new capability development is continuously slipping.
- Integration of products built by different teams reveals that incompatibilities cause many failure conditions and lead to significant out-of-cycle rework.
- Progress toward meeting milestones is unsatisfactory because unexpected rework causes cost overruns and project completion delays.

If part of the problem is due to a mismatch, in which architectural decisions fail to support business goals including agility, then part of the solution is to introduce architecture in a coherent way. This does not mean to fall back to building all of the architecture in advance of development. It is possible to introduce the concept of incremental architectural development into the agile development approach.

The solution can be described as a desired software development state that enables agile teams to quickly deliver releases that stakeholders value. When a product development starts, this desired state does not necessarily exist. The teams themselves typically define the desired state. It is their vision of the ideal development infrastructure that they would like to work with. This is a state in which platforms and frameworks, as well as tool environments and processes, exist that support efficient, independent development of user features.

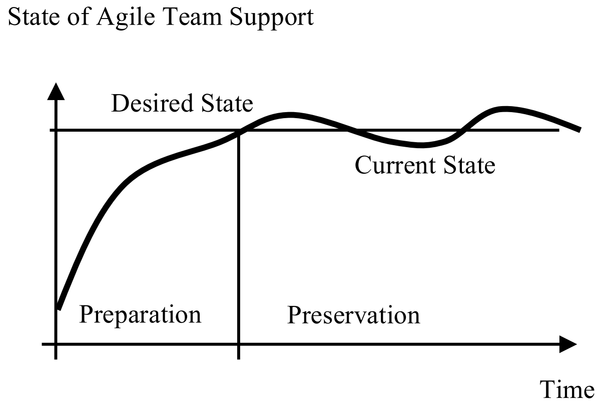

Figure 1: Infrastructure support for agile development teams over time

If the desired state does not yet exist, the agile teams first go through a preparation phase (Figure 1). The goal of this phase is to do all the preparation required to move to the desired state. During this phase, the teams can deliver releases, but they will not deliver as much value to the stakeholders as they would if they were in the desired state. This is because many tasks focus on maturing the platforms, frameworks, and tool environments.

Once they have achieved the desired state, agile teams enter the preservation phase. The goal of this phase is to maintain the desired state by dealing with technical debt, changing requirements, and new technologies. In this phase, it is critical to neither over-optimize the development infrastructure nor to quit working on it. Over-optimization incurs cost without much benefit for the

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **JUN 2012**	2. REPORT TYPE	3. DATES COVERED **00-00-2012 to 00-00-2012**

4. TITLE AND SUBTITLE **Architectural Tactics to Support Rapid and Agile Stability**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Carnegie Mellon University,Software Engineering Institute,4500 Fifth Avenue,Pittsburgh,PA,15213**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The essence of stability in software development is the ability to produce quality software with infrastructure that will meet long-term business goals. The essence of rapid and agile development is the ability to deliver capabilities quickly based on customer priorities. Stability often requires cross-functional analysis and infrastructure support that will build foundational technology for the capabilities to stand on, which takes time and resources. But today?s organizations must attend to both agility and enduring design. This article presents three tactics that support rapid and agile stability: aligning feature-based development and system decomposition, creating an architectural runway, and using matrix teams.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**6**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

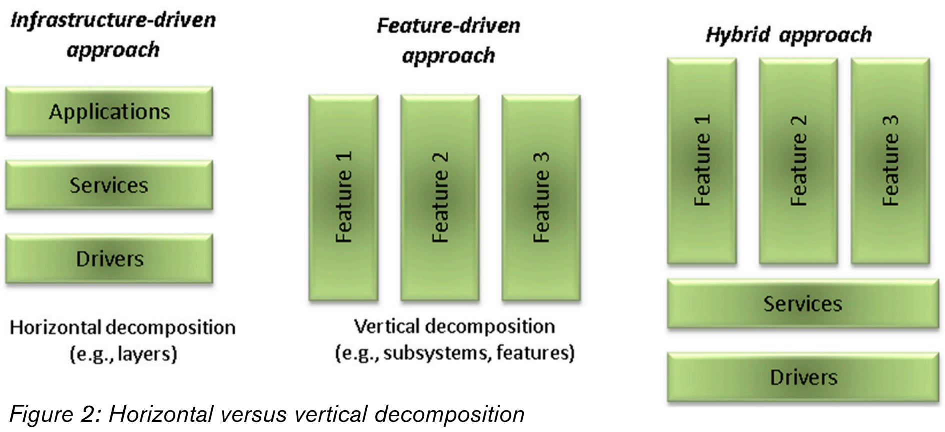

Figure 2: Horizontal versus vertical decomposition

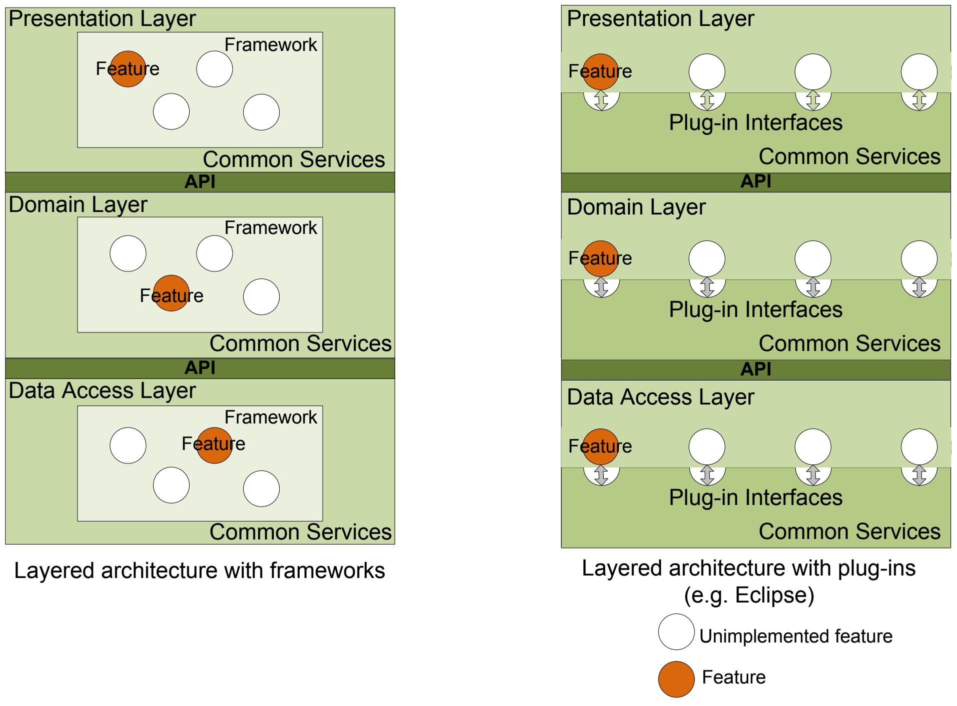

Figure 3: Layered architecture supporting feature-based development

teams (waste), while not evolving the infrastructure slows down the teams over time (also waste).

To achieve the desired productivity, different team structures must align with the desired state. Three distinct tactics can help teams move to and maintain the desired state: (1) Align feature-based development and system decomposition; (2) create an architectural runway; (3) use matrix teams. We describe these tactics in the following sections and then show how they can be used together to deliver software with agility when the agile team pays explicit attention to the underlying infrastructure to support that agility.

Aligning Feature-Based Development and System Decomposition

A common approach in agile teams is to implement a feature (or user story) in all of a system's components. This gives the team the ability to focus on something that has stakeholder value. The team controls every piece of implementation for that feature; therefore, they do not have to wait until someone outside the team has finished some required work. We call this vertical alignment because every component of the system required for realizing the feature is implemented only to the degree required by the team. System decomposition could also be horizontal based on the architectural needs of the system, focusing on common services and variability mechanisms (reuse). Figure 2 illustrates these two different approaches and how they can coexist. The goal of creating a feature-based development and system decomposition approach is to provide flexibility in aligning teams horizontally, vertically, or in combination and to ensure progress by minimizing coupling.

Although organizations create products in very different domains (from embedded systems to enterprise systems), similar architectures emerge when development teams need to support rapid and agile stability. The teams create a platform containing commonly used services and development environments either as frameworks or platform plug-ins to enable fast, feature-based development.

Figure 3 shows two generic examples of such architectures. Here we have an architecture consisting of three layers. Every layer has either a framework (left side) or a plug-in interface (right side) defined that implements the control logic of that layer. Every layer also has a collection of services that provide common functionality. To develop a feature, the agile team implements only the logic of that feature in each layer using the frameworks or plug-in interfaces. This focuses the development on what is needed for the feature implementation. The frameworks and common services already include all the logic of how to integrate new pieces of code into the system, such as by using intra-layer communication protocols. This type of architecture also minimizes the dependencies between different feature implementations so that different teams can implement features without coordination, further enhancing stability, and rapid development as users need new features.

Not every project has an existing architecture that would support feature-based development from the start. Since it is a fairly difficult task to define the appropriate APIs, common services, and plug-ins/frameworks up front, teams usually choose an evolutionary approach.

In a large multiyear client-server development project, we observed extensive use of aligning feature-based development and system decomposition to manage agile stability. Using Scrum, 25 teams participated in the development effort. Some of the teams were colocated;

other teams were located in different countries. There were teams responsible for applications, others for the platform, and others for architecture and quality assurance. In this project, the teams had a platform-oriented focus at the beginning during the preparation phase and switched to a more application-oriented focus later in the preservation phase, reflecting the change of focus in their architecture with a hybrid approach of vertical and horizontal decomposition.

Creating an Architectural Runway

An architectural runway can help provide the degree of architectural stability required to support the next iterations of development [2]. This stability is particularly important to the successful operation of multiple parallel Scrum teams. Making architectural dependencies visible allows them to be managed and for teams to be aligned with them. The runway supports the team decoupling necessary to allow independent decision-making and reduce communication and coordination overhead.

During the preparation phase, agile teams build a runway of infrastructure sufficient to support the development of features in the near future. Product development using an architectural runway most likely occurs in the preservation phase. Dean Leffingwell explains that intentional architecture is one of the key factors to successfully scale agile [2]. Building and maintaining the architectural runway puts in place a system infrastructure sufficient to allow incorporation of near-term high-priority features from the product backlog. This preparation allows the architecture to support the features without potentially creating unanticipated rework by destabilizing refactoring.

Larger systems (and teams) need longer runways. Building and rearchitecting infrastructure takes longer than a single iteration or release cycle. Delivery of planned functionality is more predictable when the infrastructure for the new features is already in place. This requires looking ahead in the planning process and investing in architecture by including infrastructure work in the present iteration that will support future features that the customer needs.

The architectural runway is not complete. The runway intentionally is not complete because of an uncertain future with changing technology and requirements. This requires continuously extending the architectural runway to support the development teams.

We observed one Scrum team that had already benefitted from an existing and proven platform. The architect of that platform was the driver and Scrum master of the development team. The team added features (vertical alignment) to the product quickly on top of the existing infrastructure while the architect, with temporarily assigned team members, implemented additional platform changes required for future features (horizontal alignment of the system into layers). The growing platform provided them with a runway sufficient to build the desired functionality for the complex embedded, real-time system environment.

Using Matrix Teams and Architecture

In its simplest instantiation, a Scrum development environment consists of a single colocated, cross-functional team with the skills, authority, and knowledge required to specify requirements and architect, design, code, and test the system. As systems grow in size and complexity, the single-team model may no longer meet development demands.

A number of different strategies can be used to scale up the overall development organization while maintaining an agile Scrum-based development approach. One approach is replication, essentially creating multiple Scrum teams with the same structure and responsibilities, sufficient to accomplish the required scope of work. This is the approach advocated by the Scrum Alliance. Some organizations scale Scrum through a hybrid approach. The hybrid approach involves Scrum team replication but also supplements the cross-functional teams with traditional functionally oriented teams. An example would be using an integration-and-test team to merge and validate code across multiple Scrum teams. (Note that Scrum purists would most likely label the hybrid approach an example of “Scrum But.”)

In general, we recognized two criteria used to organize the teams. The first criterion is organizing the teams either horizontally or vertically, assigning different teams the responsibility for either components (horizontal) or features (vertical). The second criterion is assigning the teams responsibilities according to development phases.

Aligning the teams horizontally is a good idea during the early stages of the preparation phase, while vertical alignment works well during the preservation phase. Between those two states, we find matrix structures in which the teams are aligned either horizontally or vertically while some members within those teams have the opposite responsibilities [3].

In another multiyear project, we observed two distributed Scrum teams that worked in an environment where project management and quality assurance were more waterfall oriented, causing tension because the teams delivered incremental results that were not aligned with the overall waterfall approach. When the organization decided to switch the project management and quality assurance groups to an agile approach, the architects integrated into the Scrum teams. This helped them achieve a matrix team structure to manage responsibilities for developing components and features effectively.

Applying the Tactics in Concert

Let us see how these tactics work together to provide infrastructure support for agile development over time. In Figure 4, we marked points in the state of the product development that we presented in Figure 1. These points are not exact. We use them here to give an estimate of when certain development strategies and team structures make sense.

Here we assume that when the product development starts no (or minimal) support for agile teams is available. This means there is no existing platform or frameworks, and the tool environment may not be established yet.

At the starting point (point A in Figure 4), it makes sense to organize the teams horizontally. Most of the teams’ responsibilities involve getting the supporting infrastructure to a point at which feature development can start. During this period, team members will create a rough sketch of the architecture, make technology

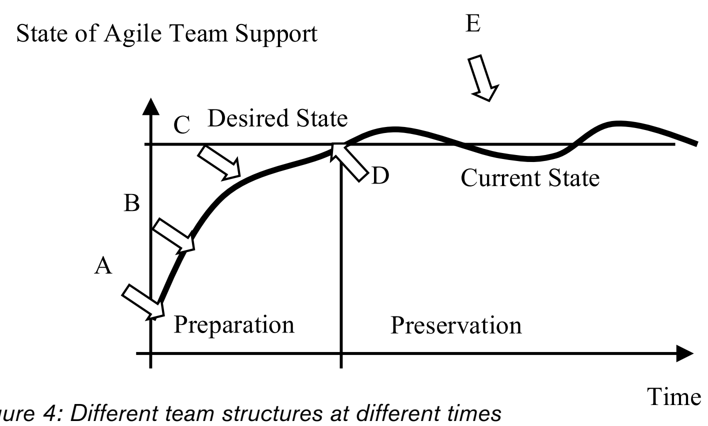

Figure 4: Different team structures at different times

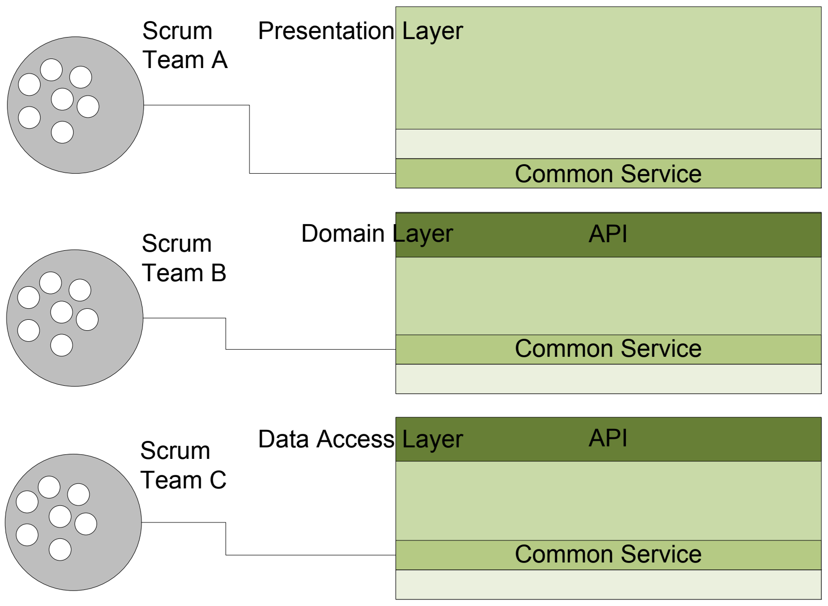

Figure 5: Layered architecture implemented with some common services and APIs

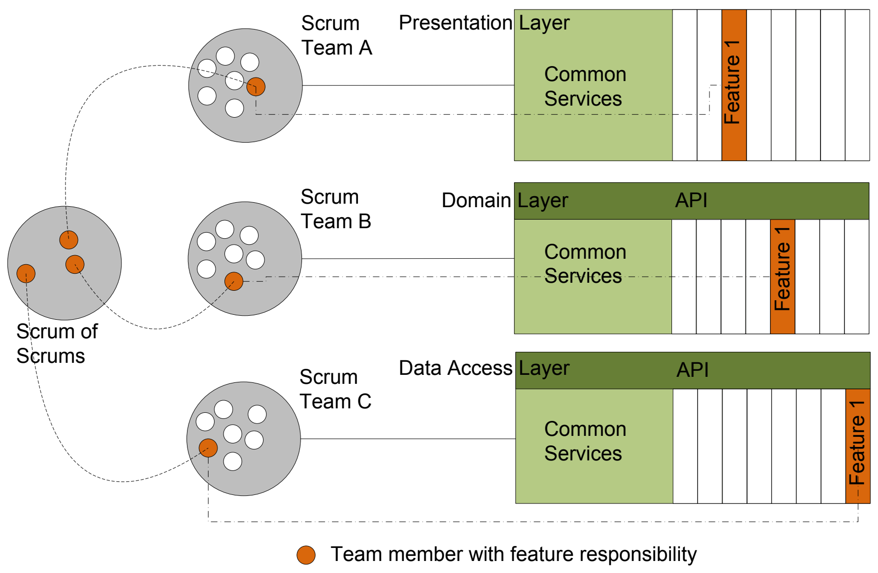

Figure 6: Feature development in parallel on top of a skeleton system; different teams assigned to layers (horizontal alignment) with some team members assigned to implement features

decisions, establish the tool environment, and so on. Typically, teams will use a small subset of basic user features (user stories) to guide the creation of the development infrastructure consisting of basic common services and APIs of the layers. Those basic features may not be implemented during this phase. Sometimes the resulting product is called a skeleton system. The result of this phase of the project is a first version of a platform that is good enough to be used to develop the first features (see Figure 5 for a notional example), as described previously in the feature-based development section.

As soon as the most important interfaces are defined, some team members can start developing features (point B in Figure 4). We now start seeing a matrix organization. During this time, most team members will still have component-oriented responsibilities. Therefore, the teams are still horizontally organized, but some team members now have the responsibility to start implementing features using the development infrastructure built so far. This pressures teams to start organizing themselves according to features and implementing them on top of the skeleton system. In a Scrum of Scrums, the team members assigned to implement features coordinate with each other to ensure on-time delivery of the features. This helps stabilize the interfaces and provides the first ideas for implementation frameworks that would support feature development (see Figure 6).

With the interfaces getting more stable, the time comes to switch most of the teams to vertical (feature-oriented) development (point C in Figure 4). In this situation, we found that some team members still had horizontal responsibilities because the development infrastructure was far from complete and teams implemented common services as well as framework and interface enhancements continuously (Figure 7).

In doing so, the teams get closer to the desired state in which they can focus on feature development (point D in Figure 4). Now the architecture has reached a level of maturity and teams have the necessary infrastructure to implement features quickly. The feature-based development aims to better manage and demonstrate end-to-end features and provides the ability to assign features to teams without too many dependencies between them. Especially in a context where there is a high number of Scrum of Scrums and many customer-facing features, this approach can help align the teams with the system structure. In one example where we observed this approach, the number of

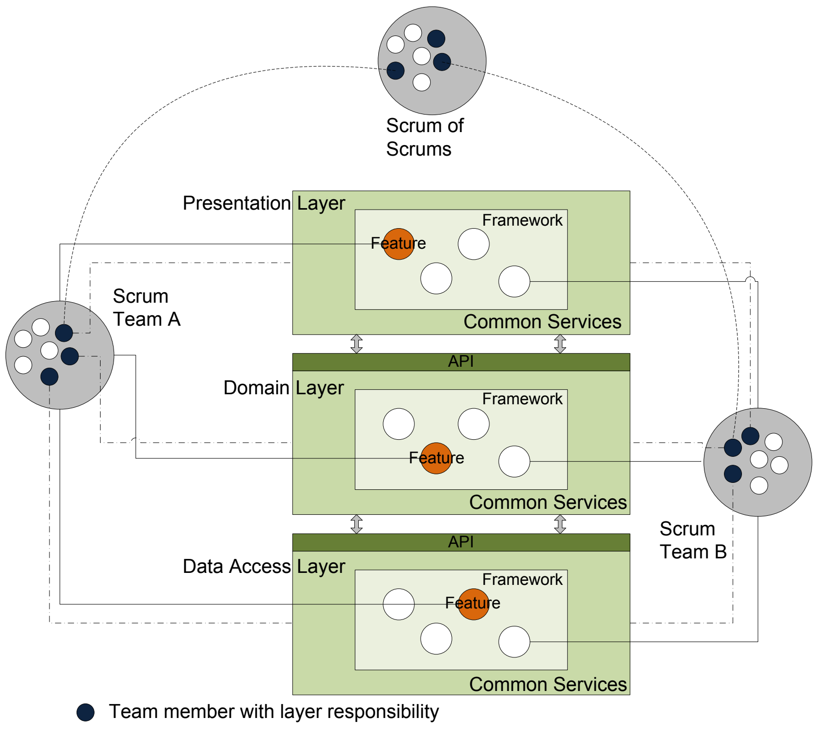

Figure 7: Different teams assigned to features (vertical alignment) with some team members assigned to keep layers and frameworks consistent

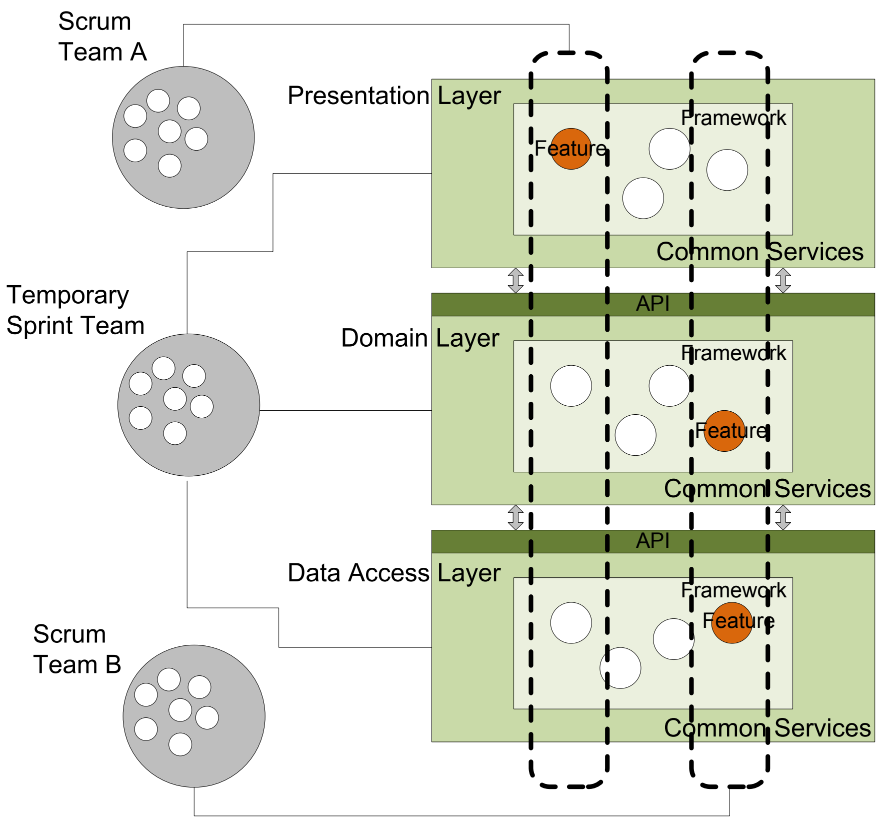

Figure 8: Different teams assigned to features (vertical alignment), with a temporary team assigned to prepare layers and frameworks for future feature development

teams participating on a Scrum of Scrums was 25; hence, the feature-based development and system decomposition approach helped separate team dependencies at the feature level.

At this point, the preservation phase starts. Few team members, if any, will have horizontal responsibilities. The goal of the preservation phase is to continue to build the next piece of the architectural runway that the system will need in the future. Every product development has to cope with changing requirements and new technologies (point E in Figure 4).

In one project we analyzed during the preservation phase, the product architect had the responsibility to look ahead and decide what the system would need in the future. He then assembled a team, and in a sprint they developed the next piece of the runway. After the sprint, the team was dissolved. Meanwhile, all the other teams were still organized vertically, developing features for their customer (Figure 8).

Takeaways

Achieving rapid and agile stability for fast yet steady development is a matter of aligning the right practices with the needs of the development effort. No one tactic can bring success to any project. The principles of both agile software development and software architecture provide improved visibility of project status and better tactics for risk management in order to create higher quality features within the required time frames and optimum use of resources. In this article, we described three tactics: aligning feature-based development and system decomposition, creating an architectural runway, and using matrix teams and architecture. Harmonious use of these tactics is critical, especially in DoD-relevant systems that must be in service for several decades, that are created by several teams and contractors, and that have changing scope due to evolving technology and emerging new needs.

Disclaimer:

This material is based upon work funded and supported by the DoD under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research, and development center.

ABOUT THE AUTHORS

Felix H. Bachmann is a senior member of the technical staff at the Software Engineering Institute (SEI) in the Architecture Centric Engineering Initiative. He is coauthor of the Attribute-Driven Design Method, a contributor to and instructor for the Architecture Tradeoff Analysis Method® Evaluator Training, and coauthor of Documenting Software Architectures: Views and Beyond. Before joining the SEI, he was a software engineer at Robert Bosch GmbH in Corporate Research for small and large embedded systems.

Software Engineering Institute
4500 Fifth Avenue
Pittsburgh, PA
E-mail: fb@sei.cmu.edu
Phone: 412-268-6194

ABOUT THE AUTHORS (continued)

Robert L. Nord is a senior member of the technical staff at SEI and works to develop and communicate effective methods and practices for software architecture. He is coauthor of the practitioner-oriented books Applied Software Architecture and Documenting Software Architectures: Views and Beyond and lectures on architecture-centric approaches.

E-mail: rn@sei.cmu.edu
Phone: 412-268-1705

Ipek Ozkaya is a senior member of the technical staff at SEI and works to develop empirical methods for improving software development efficiency and system evolution with a focus on software architecture practices, software economics, and requirements management. Her latest publications include multiple articles on these subjects focusing on agile architecting, dependency management, and architectural technical debt. Ozkaya serves on the advisory board of IEEE Software.

E-mail: ozkaya@sei.cmu.edu
Phone: 412-268-3551

REFERENCES

1. Brown, N., R. Nord, and I. Ozkaya. "Enabling Agility Through Architecture." CrossTalk 23.6 (2010): 12-17.
2. Leffingwell, D. Scaling Software Agility. Upper Saddle River, NJ: Addison-Wesley, 2007.
3. Reinertsen, D. G. Managing the Design Factory: A Product Developer's Toolkit. New York: The Free Press, 1997.

NAVAIR Vision Statement:
"Sailors and Marines, Armed and Operating with Confidence"

Because we develop, deliver, and sustain aircraft, weapons, and systems—on time, on cost, with proven capability and reliability—so they cost effectively succeed in every mission and return home safely.

NAVAIR Process Resource Team (PRT)
(760) 939-6226